Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 5 网络数据收集和分析 Web Data Collection and Analysis

内容：

- 数据分析实战：本地数据和Web数据

实践：

- Web数据获取
- 股票金融数据分析
- 可视化绘图
- 不同文件格式的磁盘文件读取和保存
- 文件格式（csv，excel，json，html等）

上节课讲述了从本地和Web获取数据，并进行数据的预处理、分析、可视化和磁盘保存。

本章目录：

- 1) 读入本地磁盘数据，并进行数据分析，绘图
- 2) 获取Web数据（股票数据），并进行股票数据的分析和处理，保存磁盘

In [1]:
```
# 必要准备工作：导入库，配置环境等
#from __future__ import division
#import os, sys

# 导入库并为库起个别名
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

本地数据集1：餐馆小费

tips.csv 是个关于餐馆小费记录的数据，包含七个字段（total_bill，tip，sex，smoker，day，time，size），共计244条记录。

- 磁盘读入csv格式文件转为pd数据结构
- 对数据分析（缺失值-填充，清理，汇总描述，可视化绘图）
- 数据相关性分析
- 数据分组聚合

In [2]:
```
import pandas as pd
tips = 'data/tips.csv'
data = pd.read_csv(tips)  #默认header='infer'，推导第一行是header，小费记录始于第二行
print len(data) #数据记录数量
data.head()  #预览最前5行记录
#data.tail() #预览最后5行记录
```

244

Out[2]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [3]:
```
data.describe() #数据的汇总描述
```

Out[3]:

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

In [4]:
```
stat = data.describe() #数据的基本统计量
#重点：可以自己添加统计量信息
stat.loc['range'] = stat.loc['max'] - stat.loc['min']  #极差 range
stat.loc['var'] = stat.loc['std'] / stat.loc['mean']  #变异系数，标准差/均值的离中趋势
stat.loc['dis'] =  stat.loc['75%'] - stat.loc['25%'] #四分位数间距（极差）
stat
```

Out[4]:

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000

75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000
range	47.740000	9.000000	5.000000
var	0.449936	0.461478	0.370125
dis	10.780000	1.562500	1.000000

In [5]:
```
#数据的其他统计量
#数据的分布统计
data.median()   #数据的中位数
data.mode() #数据的众数
data.quantile(0.1)   #数据的百分位数 data.quantile(q=0.5)

#数据的离中趋势度量
data.skew() #数据的偏度
data.kurt() #数据的峰度

#数据列之间的相关度量
data.cov() #数据的协方差矩阵
data.corr() #数据的Pearson相关系数矩阵
```

Out[5]:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

In [6]:
```
data.columns
```

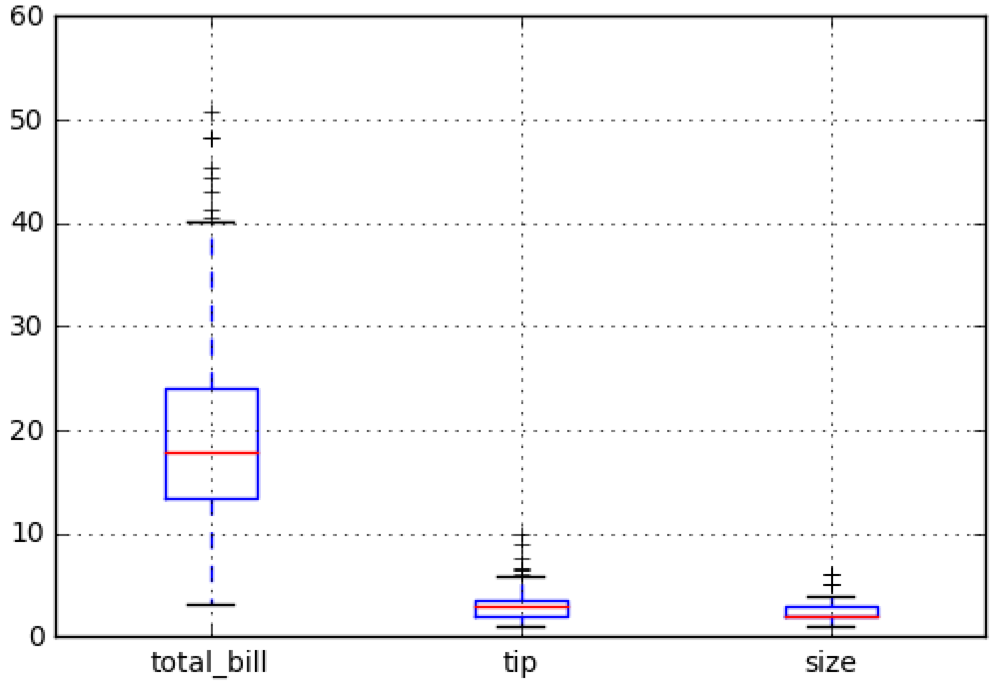
Out[6]:
```
Index([u'total_bill', u'tip', u'sex', u'smoker', u'day', u'time', u'size
'], dtype='object')
```

In [7]:
```
data.columns.names   #此时每个列没有name
```

Out[7]:
```
FrozenList([None])
```

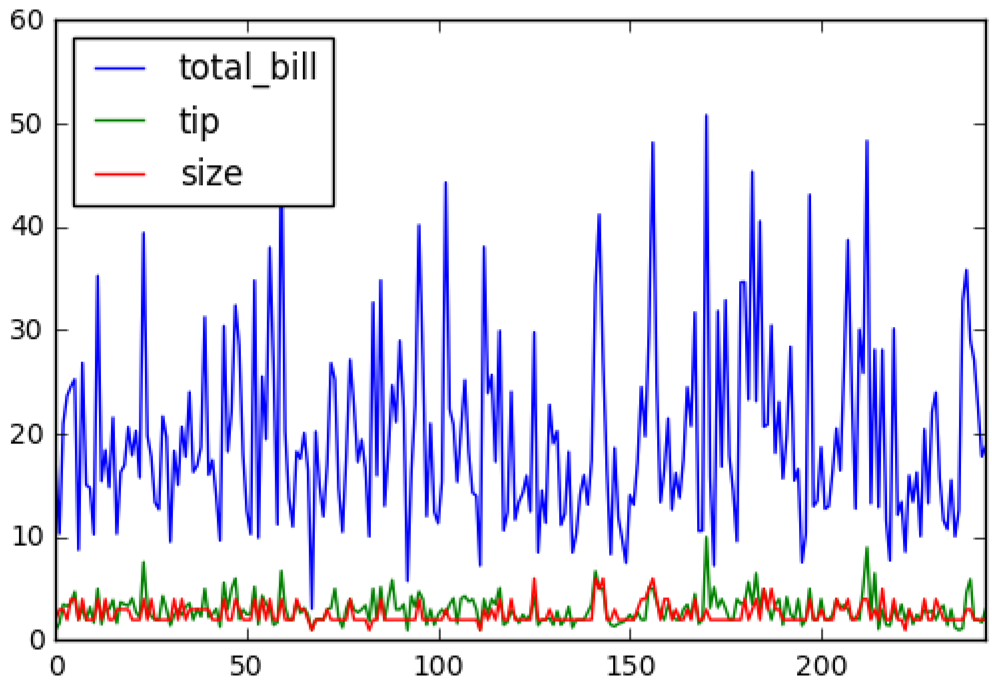
In [8]:
```
#启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

data.boxplot(return_type='axes')   #画盒图，直接使用DataFrame的方法
#data.boxplot() #画盒图，直接使用DataFrame的方法，需要屏蔽warning
#data[['tip','size']].boxplot(return_type='axes') #只对两个列画盒图
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x9d066d8>
```



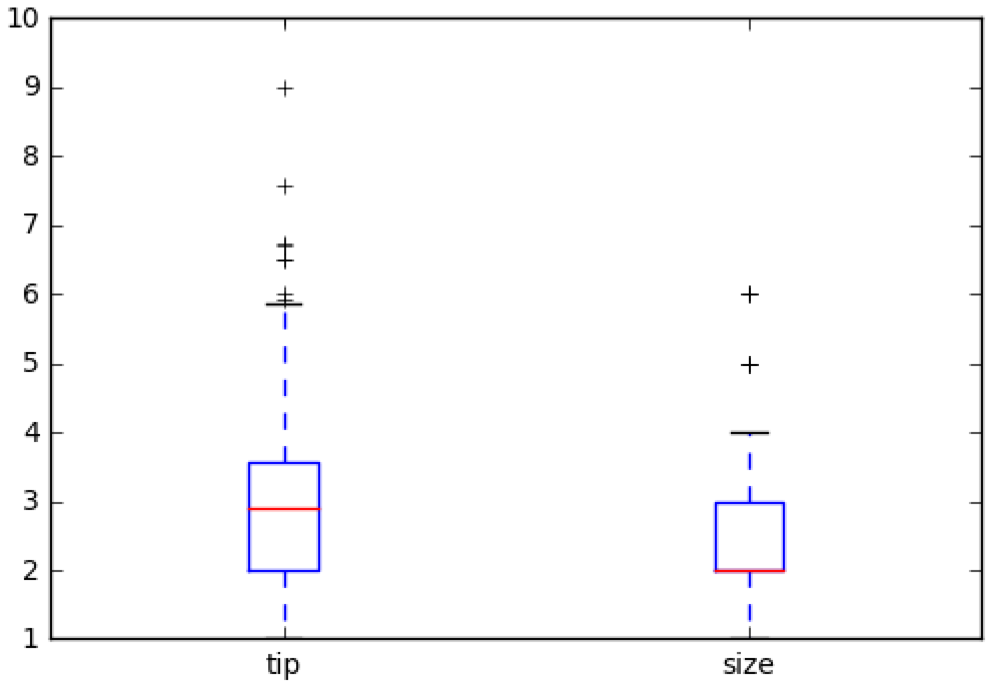
In [9]:
```
data.plot.line()  #线图
#data.plot.hist() #直方图
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x9ee9e48>



In [10]:
```
#几种盒图绘图
#data[['tip','size']].boxplot() # 盒图1, 同前面
#data[['tip','size']].plot(kind='box') # 盒图2
data[['tip','size']].plot.box()  # 盒图3
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0xa1ef8d0>



In [11]:
```
# 其他多种图类型, tip, size, total_bill
#data['tip'].plot.line()  # 线图 1
#data['tip'].plot(kind='line')  # 线图 2
#data['tip'].plot.hist()  # 直方图
#data['tip'].plot.kde()  # 密度图1 'kde' : Kernel Density Estimation plot
#data['tip'].plot.density() # 密度图1 density,同上
#data['tip'].plot.pie() # 饼图
```

In [12]:
```
# 相关性分析
data.corr()    # method : {'pearson', 'kendall', 'spearman'}, 默认pearson
```

Out[12]:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

In [13]:
```
data.corr(method='kendall')    # method : {'pearson', 'kendall', 'spearman'}
```

Out[13]:

	total_bill	tip	size
total_bill	1.000000	0.517181	0.484342
tip	0.517181	1.000000	0.378185
size	0.484342	0.378185	1.000000

In [14]:
```
data.head()
```

Out[14]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3

3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

思考： 数据分组进一步考虑：消费与date（周一周末）是否有关？是否与time（中餐晚餐）有关？

怎么做？

- 把day和time两个列转为行索引的外层和内层
- DataFrame的set_index函数会将其一个或多个列转换为行索引，并创建一个新的DataFrame。

In [15]:

```
# 考虑bill与date（周一周末）是否有关？是否与time（中餐晚餐）有关？
# 把day和time两个列转为行索引的外层和内层
# DataFrame的set_index函数会将其一个或多个列转换为行索引，并创建一个新的DataFrame。
data2 = data.set_index(['day', 'time'])
data2.head()
```

Out[15]:

		total_bill	tip	sex	smoker	size
day	**time**					
Sun	**Dinner**	16.99	1.01	Female	No	2
	Dinner	10.34	1.66	Male	No	3
	Dinner	21.01	3.50	Male	No	3
	Dinner	23.68	3.31	Male	No	2
	Dinner	24.59	3.61	Female	No	4

In [16]:

```
# 选取周日Sun的消费统计汇总情况
data2.ix['Sun'].describe()
```

Out[16]:

	total_bill	tip	size
count	76.000000	76.000000	76.000000
mean	21.410000	3.255132	2.842105
std	8.832122	1.234880	1.007341
min	7.250000	1.010000	2.000000
25%	14.987500	2.037500	2.000000
50%	19.630000	3.150000	2.000000
75%	25.597500	4.000000	4.000000
max	48.170000	6.500000	6.000000

In [17]:

```
# 选取周日Sun晚餐Dinner的消费统计汇总情况
data2.ix['Sun'].ix['Dinner'].describe()
#data2.ix['Sun'].ix['Lunch'].describe() # 周日没有Lunch消费的记录
```

Out[17]:

	total_bill	tip	size
count	76.000000	76.000000	76.000000
mean	21.410000	3.255132	2.842105

std	8.832122	1.234880	1.007341
min	7.250000	1.010000	2.000000
25%	14.987500	2.037500	2.000000
50%	19.630000	3.150000	2.000000
75%	25.597500	4.000000	4.000000
max	48.170000	6.500000	6.000000

In [18]:

```
#对比工作日周五的午餐和晚餐消费均值
print data2.ix['Fri'].ix['Lunch'].mean() #选取周五Fri午餐Lunch的消费统计汇总情况
print data2.ix['Fri'].ix['Dinner'].mean() #选取周五Fri晚餐Dinner的消费统计汇总情况
```

```
total_bill    12.845714
tip            2.382857
size           2.000000
dtype: float64
total_bill    19.663333
tip            2.940000
size           2.166667
dtype: float64
```

In [19]:

```
#对比周五到周日的消费均值
print data2.ix['Fri']['total_bill'].mean()
print data2.ix['Sat']['total_bill'].mean()
print data2.ix['Sun']['total_bill'].mean()
```

```
17.1515789474
20.4413793103
21.41
```

In [20]:

```
#交换索引（行索引的内层和外层索引交换）
data3 = data2.swaplevel(0,1)  #交换索引后返回新的data3
data3.tail()
```

Out[20]:

		total_bill	tip	sex	smoker	size
time	**day**					
Dinner	**Sat**	29.03	5.92	Male	No	3
	Sat	27.18	2.00	Female	Yes	2
	Sat	22.67	2.00	Male	Yes	2
	Sat	17.82	1.75	Male	No	2
	Thur	18.78	3.00	Female	No	2

In [21]:

```
#比较午餐Lunch和晚餐Dinner的消费统计汇总情况
print data3.ix['Dinner'].describe()
print data3.ix['Lunch'].describe()
```

```
       total_bill         tip        size
count  176.000000  176.000000  176.000000
mean    20.797159    3.102670    2.630682
std      9.142029    1.436243    0.910241
```

```
min       3.070000    1.000000    1.000000
25%      14.437500    2.000000    2.000000
50%      18.390000    3.000000    2.000000
75%      25.282500    3.687500    3.000000
max      50.810000   10.000000    6.000000
       total_bill        tip        size
count   68.000000  68.000000  68.000000
mean    17.168676   2.728088   2.411765
std      7.713882   1.205345   1.040024
min      7.510000   1.250000   1.000000
25%     12.235000   2.000000   2.000000
50%     15.965000   2.250000   2.000000
75%     19.532500   3.287500   2.000000
max     43.110000   6.700000   6.000000
```

其实，我们不必进行上面的操作，因为pandas提供了非常方便的groupby分组操作

groupby分组操作

pandas的DataFrame有groupby操作，可以非常方便对数据分组。不需要将多个列索引转换为行索引的情况下，可以直接对数据进行分组分析计算。

- df.groupby(['col2', 'col3']) # 首先，按照col2和col3的不同值进行分组
- df['col1'].describe() # 然后，统计col1的汇总情况

In [22]:

```
#添加“小费占总额百分比”列
data['tip_pct'] = data['tip'] / data['total_bill']
data[:6]
```

Out[22]:

	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	16.99	1.01	Female	No	Sun	Dinner	2	0.059447
1	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
3	23.68	3.31	Male	No	Sun	Dinner	2	0.139780
4	24.59	3.61	Female	No	Sun	Dinner	4	0.146808
5	25.29	4.71	Male	No	Sun	Dinner	4	0.186240

In [23]:

```
#分组统计
data.groupby(['sex','smoker']).count() #统计不同性别和是否抽烟的数量
```

Out[23]:

		total_bill	tip	day	time	size	tip_pct
sex	**smoker**						
Female	**No**	54	54	54	54	54	54
	Yes	33	33	33	33	33	33
Male	**No**	97	97	97	97	97	97
	Yes	60	60	60	60	60	60

In [24]:

```
#分组统计
data.groupby(['day','time']).count()   #统计不同天和不同餐时的数量
```

Out[24]:

		total_bill	tip	sex	smoker	size	tip_pct
day	time						
Fri	Dinner	12	12	12	12	12	12
	Lunch	7	7	7	7	7	7
Sat	Dinner	87	87	87	87	87	87
Sun	Dinner	76	76	76	76	76	76
Thur	Dinner	1	1	1	1	1	1
	Lunch	61	61	61	61	61	61

In [25]:

```
#统计不同天和时间的平均情况
data.groupby(['day','time']).mean()
```

Out[25]:

		total_bill	tip	size	tip_pct
day	time				
Fri	Dinner	19.663333	2.940000	2.166667	0.158916
	Lunch	12.845714	2.382857	2.000000	0.188765
Sat	Dinner	20.441379	2.993103	2.517241	0.153152
Sun	Dinner	21.410000	3.255132	2.842105	0.166897
Thur	Dinner	18.780000	3.000000	2.000000	0.159744
	Lunch	17.664754	2.767705	2.459016	0.161301

In [26]:

```
#只统计不同天和时间的tip平均情况
data['tip'].groupby([data['day'], data['time']]).mean()
```

Out[26]:

```
day   time
Fri   Dinner    2.940000
      Lunch     2.382857
Sat   Dinner    2.993103
Sun   Dinner    3.255132
Thur  Dinner    3.000000
      Lunch     2.767705
Name: tip, dtype: float64
```

In [27]:

```
#比较不同性别在不同天的午餐Lunch和晚餐Dinner的平均小费情况
data['tip'].groupby([data['sex'], data['time']]).mean()
```

Out[27]:

```
sex     time
Female  Dinner    3.002115
        Lunch     2.582857
Male    Dinner    3.144839
        Lunch     2.882121
Name: tip, dtype: float64
```

In [28]:

```
#综合比较不同性别在周末午餐Lunch和晚餐Dinner的平均消费情况
data.groupby(['sex', 'time']).mean()
```

Out[28]:

		total_bill	tip	size	tip_pct
sex	time				

Female	**Dinner**	19.213077	3.002115	2.461538	0.169322
	Lunch	16.339143	2.582857	2.457143	0.162285
Male	**Dinner**	21.461452	3.144839	2.701613	0.155407
	Lunch	18.048485	2.882121	2.363636	0.166083

In [29]:
```
# 分组统计不同性别给出小费的比例情况
grouped = data.groupby(['sex'])
grouped.mean()
```

Out[29]:

	total_bill	tip	size	tip_pct
sex				
Female	18.056897	2.833448	2.459770	0.166491
Male	20.744076	3.089618	2.630573	0.157651

In [30]:
```
# 不同性别给小费比例的均值
grouped['tip_pct'].mean()
```

Out[30]:
```
sex
Female    0.166491
Male      0.157651
Name: tip_pct, dtype: float64
```

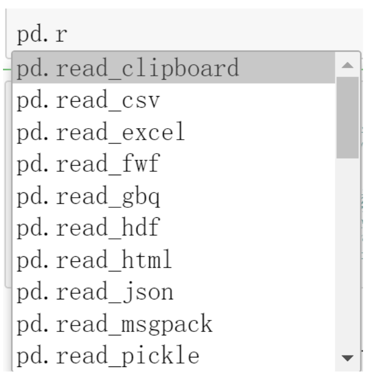
In [31]:
```
# 首先，根据sex和smoker对tips进行分组
grouped = data['tip_pct'].groupby([data['sex'],data['smoker']])
grouped.mean()
```

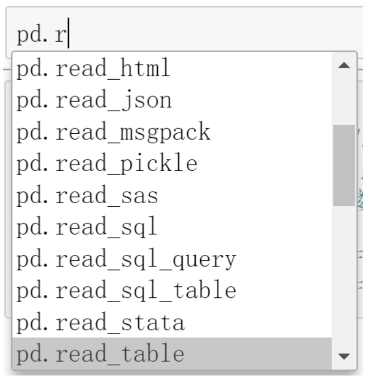
Out[31]:
```
sex     smoker
Female  No        0.156921
        Yes       0.182150
Male    No        0.160669
        Yes       0.152771
Name: tip_pct, dtype: float64
```

小作业1-a

- 考虑并分析其他特征或特征组合，如性别、是否抽烟、就餐人数等与消费量和消费习惯等的情况
- 对各种分析的可视化展示

pandas可以读入的数据文件格式包括：





pandas读入多种数据格式：

- csv, excel

- html, json

- pickle

- 剪贴板

- 数据库**, sql, hdf**

- SASXport，**Google BigQuery**等

- 通用表格**, table**

本地数据集2：股票时间序列数据

2 - stock_px.csv 是个关于股票股价记录的时间序列数据，包含9个字段（AA，AAPL，GE，IBM，JNJ，MSFT，PEP，SPX，XOM），时间从1990/2/1到2011/10/14，共计5472条记录。

美铝公司[AA]，苹果公司[AAPL]，通用电气[GE]，微软[MSFT]，强生[JNJ]，百事[PEP]，美国标准普尔500指数 (SPX)，埃克森美孚[XOM]

In [32]:
```
import pandas as pd
import numpy as np
from datetime import datetime

f = 'data/stock_px.csv'
data = pd.read_csv(f, index_col='date')  # 使用date列作为行索引
data.index = pd.to_datetime(data.index)  # 将字符串索引转换成时间索引
print len(data) # 数据记录数量
data.head()  # 预览最前5行记录
data.tail()  # 预览最后5行记录
```

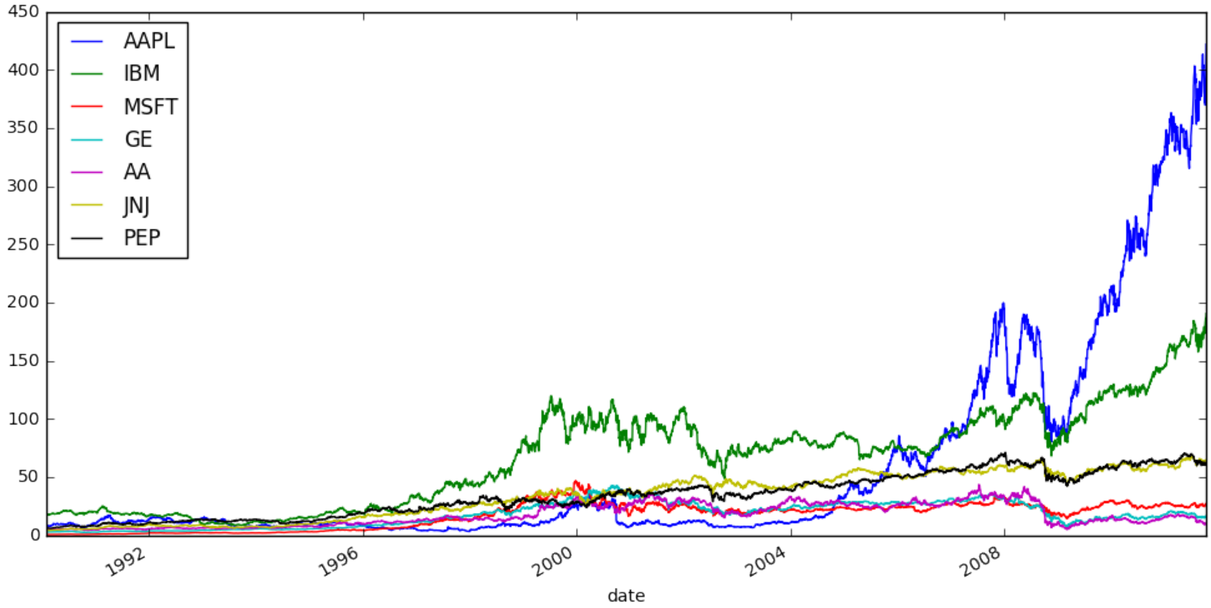
5472

Out[32]:

	AA	AAPL	GE	IBM	JNJ	MSFT	PEP	SPX	XOM
date									
2011-10-10	10.09	388.81	16.14	186.62	64.43	26.94	61.87	1194.89	76.28
2011-10-11	10.30	400.29	16.14	185.00	63.96	27.00	60.95	1195.54	76.27
2011-10-12	10.05	402.19	16.40	186.12	64.33	26.96	62.70	1207.25	77.16
2011-10-13	10.10	408.43	16.22	186.82	64.23	27.18	62.36	1203.66	76.37
2011-10-14	10.26	422.00	16.60	190.53	64.72	27.27	62.24	1224.58	78.11

In [33]:
```
plt.rc('figure', figsize=(12,6))
data[['AAPL','IBM','MSFT','GE','AA','JNJ','PEP']].plot.line() #绘曲线图
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0xdb04b70>



In [34]:
```
#重点了解苹果股票的基本统计量
data['AAPL'].describe()
```

Out[34]:
```
count    5472.000000
mean       57.119313
std        88.670423
min         3.230000
25%         8.760000
50%        11.990000
75%        68.017500
max       422.000000
Name: AAPL, dtype: float64
```

In [35]:
```
#苹果股票的其他统计量
print data['AAPL'].median()   #数据的中位数
print data['AAPL'].mode() #数据的众数
print data['AAPL'].quantile(0.1)   #数据的百分位数 data.quantile(q=0.5)
print data['AAPL'].skew() #数据的偏度
print data['AAPL'].kurt() #数据的峰度
```

```
11.99
0    9.28
dtype: float64
6.51
2.11068944167
3.7246920435
```

In [36]:
```
#各股票之间的相关统计量
#data.cov() #数据的协方差矩阵
data.corr() #数据的Pearson相关系数矩阵
```

Out[36]:

	AA	AAPL	GE	IBM	JNJ	MSFT	PEP	SPX	XO
AA	1.000000	0.101313	0.916804	0.600211	0.685752	0.776796	0.634679	0.846861	0.5
AAPL	0.101313	1.000000	0.142381	0.749037	0.651564	0.423274	0.741942	0.410813	0.7
GE	0.916804	0.142381	1.000000	0.681659	0.717824	0.875398	0.652904	0.935598	0.5
IBM	0.600211	0.749037	0.681659	1.000000	0.902894	0.871615	0.885029	0.835484	0.8
JNJ	0.685752	0.651564	0.717824	0.902894	1.000000	0.846906	0.970478	0.845401	0.9

MSFT	0.776796	0.423274	0.875398	0.871615	0.846906	1.000000	0.781791	0.949715	0.7
PEP	0.634679	0.741942	0.652904	0.885029	0.970478	0.781791	1.000000	0.816477	0.9
SPX	0.846861	0.410813	0.935598	0.835484	0.845401	0.949715	0.816477	1.000000	0.7
XOM	0.567025	0.781369	0.581171	0.855195	0.925600	0.730557	0.964278	0.761077	1.0

In [37]:
```
data.head()
```
Out[37]:

	AA	AAPL	GE	IBM	JNJ	MSFT	PEP	SPX	XOM
date									
1990-02-01	4.98	7.86	2.87	16.79	4.27	0.51	6.04	328.79	6.12
1990-02-02	5.04	8.00	2.87	16.89	4.37	0.51	6.09	330.92	6.24
1990-02-05	5.07	8.18	2.87	17.32	4.34	0.51	6.05	331.85	6.25
1990-02-06	5.01	8.12	2.88	17.56	4.32	0.51	6.15	329.66	6.23
1990-02-07	5.04	7.77	2.91	17.93	4.38	0.51	6.17	333.75	6.33

In [38]:
```
data.tail()
```
Out[38]:

	AA	AAPL	GE	IBM	JNJ	MSFT	PEP	SPX	XOM
date									
2011-10-10	10.09	388.81	16.14	186.62	64.43	26.94	61.87	1194.89	76.28
2011-10-11	10.30	400.29	16.14	185.00	63.96	27.00	60.95	1195.54	76.27
2011-10-12	10.05	402.19	16.40	186.12	64.33	26.96	62.70	1207.25	77.16
2011-10-13	10.10	408.43	16.22	186.82	64.23	27.18	62.36	1203.66	76.37
2011-10-14	10.26	422.00	16.60	190.53	64.72	27.27	62.24	1224.58	78.11

股票的时间序列数据分析

除了前面常用的统计、汇总、分组、可视化分析，对于股票数据，还可以进行更复杂的数据分析任务：

- 根据每天的收盘价返回对数收益率

In [39]:
```
# 只取出苹果股票分析
df = pd.DataFrame(data['AAPL'],index=data.index, columns=['AAPL'])   # data['AAPL']只是个Series
df.tail()
```
Out[39]:

	AAPL
date	
2011-10-10	388.81
2011-10-11	400.29

2011-10-12	402.19
2011-10-13	408.43
2011-10-14	422.00

In [40]:
```
# 更复杂的数据分析任务：根据每天的收盘价返回对数收益率
# 首先添加包含对应信息的列，生成一个新的列，
# 然后中所有股价上进行循环，逐步计算单个对数收益率值
df['Return'] = 0.0
for i in range(1, len(df)):
    df['Return'][i] = np.log(df['AAPL'][i] / df['AAPL'][i-1])
df.tail()
```

Out[40]:

	AAPL	Return
date		
2011-10-10	388.81	0.050128
2011-10-11	400.29	0.029098
2011-10-12	402.19	0.004735
2011-10-13	408.43	0.015396
2011-10-14	422.00	0.032685

In [41]:
```
# 也可以使用向量化代码，在不使用循环的情况下得到相同的结果，即shift方法
df['Return2'] = np.log(df['AAPL'] / df['AAPL'].shift(1))
df[['AAPL', 'Return', 'Return2']].tail()
# 最后面两列的值相同：更紧凑和更容易理解的代码，而且是更快速的替代方案
```

Out[41]:

	AAPL	Return	Return2
date			
2011-10-10	388.81	0.050128	0.050128
2011-10-11	400.29	0.029098	0.029098
2011-10-12	402.19	0.004735	0.004735
2011-10-13	408.43	0.015396	0.015396
2011-10-14	422.00	0.032685	0.032685

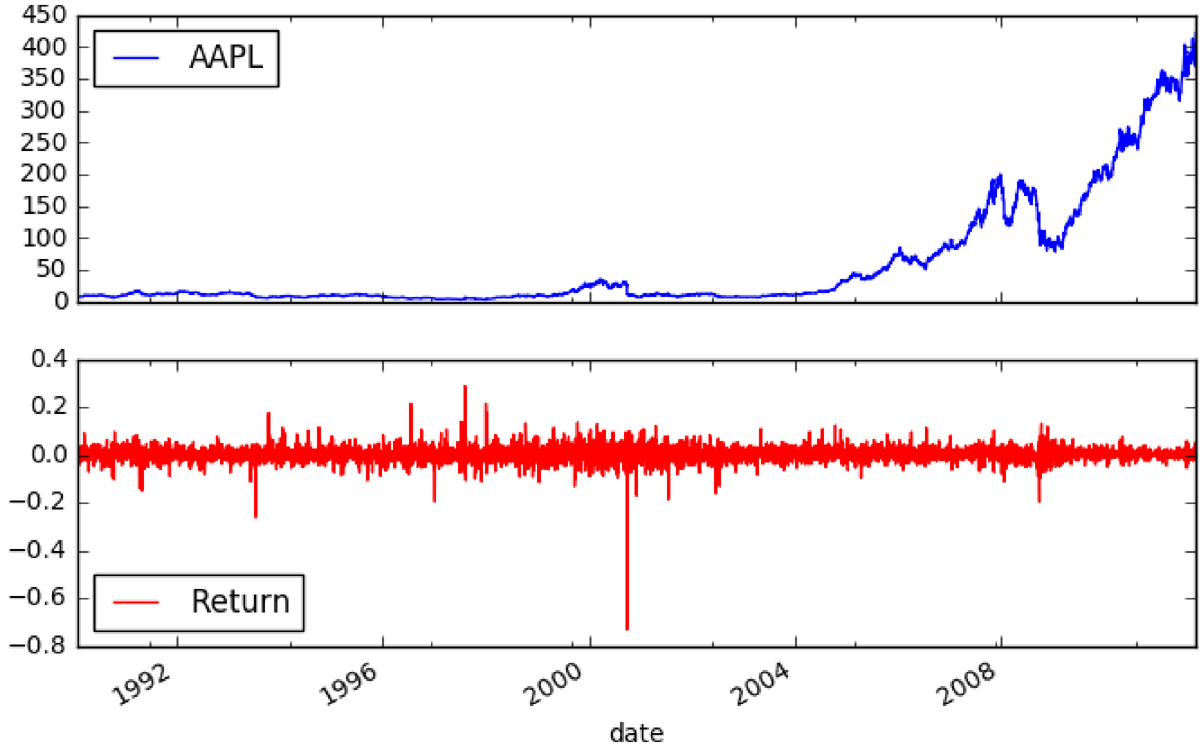
In [42]:
```
# 目前，一个对数收益率数据列就足够了，可以删除另一个列
del df['Return2'] # 删除列
df.tail()
```

Out[42]:

	AAPL	Return
date		
2011-10-10	388.81	0.050128
2011-10-11	400.29	0.029098
2011-10-12	402.19	0.004735
2011-10-13	408.43	0.015396
2011-10-14	422.00	0.032685

In [43]:
```
#绘图更好地概览股价和波动率变化
df[['AAPL', 'Return']].plot(subplots=True, style=['b','r'], figsize=(8,5
))
```

Out[43]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000000000DAEDE
F0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000000000E31B2
78>], dtype=object)
```



In [44]:
```
#技术型股票交易者可能对移动平均值（即趋势）更感兴趣，
#移动平均值很容易使用pandas的rolling_mean计算
df['42d'] = pd.rolling_mean(df['AAPL'], window = 42)
df['252d'] = pd.rolling_mean(df['AAPL'], window = 252)
df[['AAPL', '42d', '252d']].tail()
```

```
C:\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: FutureWarning: p
d.rolling_mean is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=42,center=False).mean()
  app.launch_new_instance()
C:\Anaconda2\lib\site-packages\ipykernel\__main__.py:4: FutureWarning: p
d.rolling_mean is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=252,center=False).mean()
```

Out[44]:

	AAPL	42d	252d
date			
2011-10-10	388.81	384.502381	346.165278
2011-10-11	400.29	385.135476	346.569048
2011-10-12	402.19	385.735476	346.974008
2011-10-13	408.43	386.331190	347.395119
2011-10-14	422.00	387.319762	347.820754

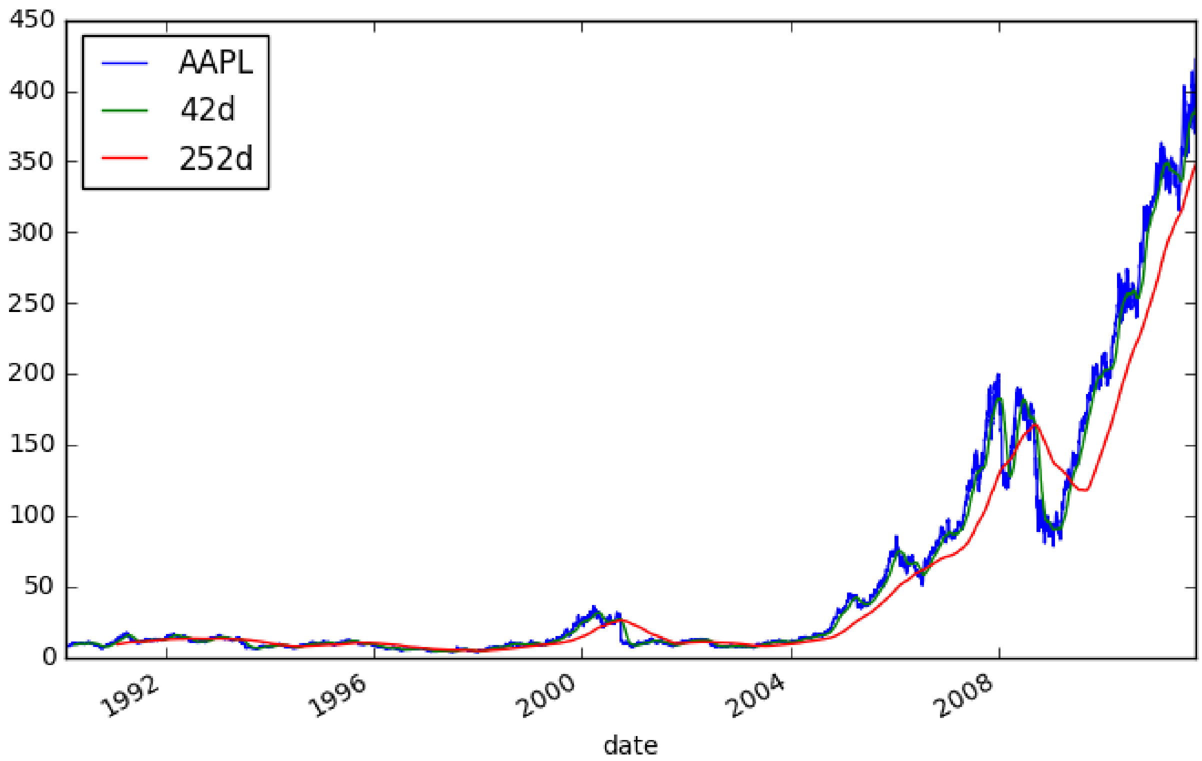
In [45]:
```
df[['AAPL', '42d', '252d']].head() #对于后两列，前面的数据为空
```

Out[45]:

	AAPL	42d	252d
date			
1990-02-01	7.86	NaN	NaN
1990-02-02	8.00	NaN	NaN
1990-02-05	8.18	NaN	NaN
1990-02-06	8.12	NaN	NaN
1990-02-07	7.77	NaN	NaN

In [46]:
```
#包含两种趋势的典型股价图表绘图
df[['AAPL', '42d', '252d']].plot(figsize=(8,5))
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0xeee74e0>



In [47]:
```
#期权交易者更喜欢的话题，对数收益率的移动历史标准差--即移动历史波动率
import math
df['Mov_Vol'] = pd.rolling_mean(df['Return'], window=252) * math.sqrt(25
2)
df['Mov_Vol'].tail() #Mov_Vol列的数据在前面252行为空
```

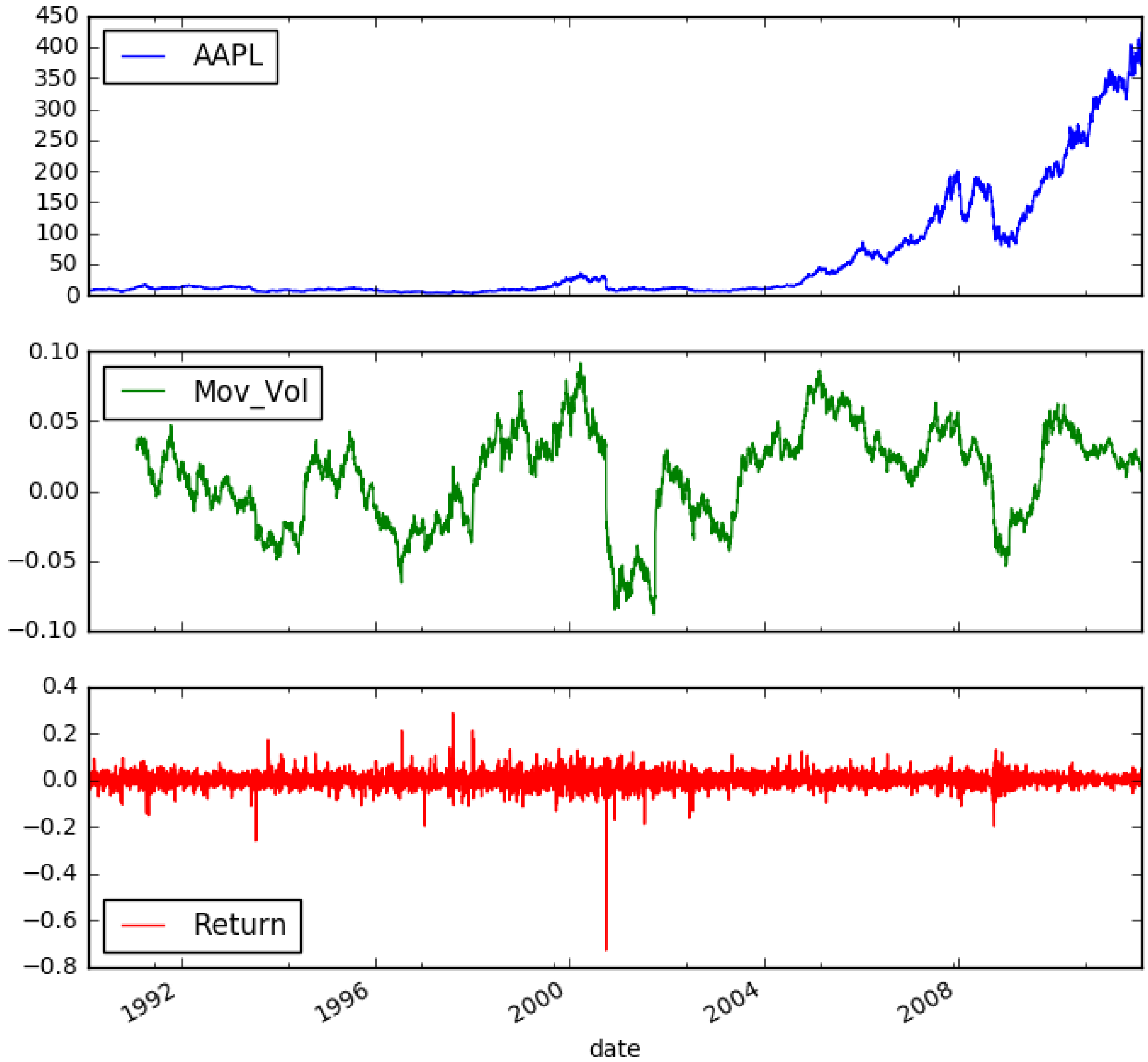
```
C:\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: FutureWarning: p
d.rolling_mean is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=252,center=False).mean()
  app.launch_new_instance()
```

Out[47]:
```
date
2011-10-10    0.017317
2011-10-11    0.018475
2011-10-12    0.018437
```

```
2011-10-13    0.018953
2011-10-14    0.018474
Name: Mov_Vol, dtype: float64
```

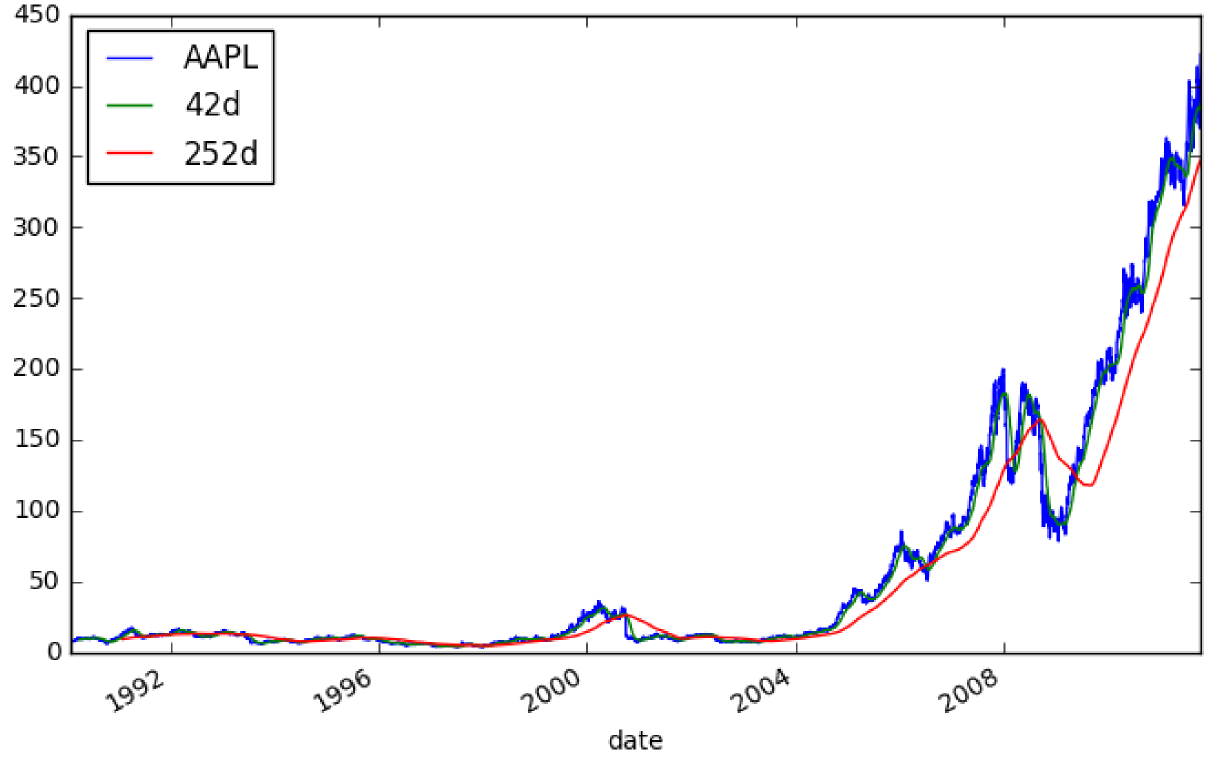
In [48]:
```
#杠杆效应假设，说明市场下跌时历史移动波动率倾向于升高，而在市场上涨时波动率下降
df[['AAPL', 'Mov_Vol', 'Return']].plot(subplots=True, style=['b','g','r'
], figsize=(8,8))
```

Out[48]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000000000F2EDB
E0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000000000F4702
B0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000000000F640B
00>], dtype=object)
```



In [49]:
```
#包含两种趋势的典型股价图表绘图
df[['AAPL', '42d', '252d']].plot(figsize=(8,5))
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0xf2edc50>



In [50]:
```
df.tail()
```

Out[50]:

	AAPL	Return	42d	252d	Mov_Vol
date					
2011-10-10	388.81	0.050128	384.502381	346.165278	0.017317
2011-10-11	400.29	0.029098	385.135476	346.569048	0.018475
2011-10-12	402.19	0.004735	385.735476	346.974008	0.018437
2011-10-13	408.43	0.015396	386.331190	347.395119	0.018953
2011-10-14	422.00	0.032685	387.319762	347.820754	0.018474

保存数据

现在我们想把对苹果股票的分析数据保存下来，在以后的分析中继续使用。pandas的DataFrame的保存数据类型，参考前面表格中的读入数据类型。

In [51]:
```
#为了以后更容易导入数据，我们生成一个新的csv数据文本，并将所有数据行写入新文件
out_file = open('data/aapl.csv', 'w')
df.to_csv(out_file)
out_file.close()
```

In [52]:
```
# 使用Yahoo Finance的API获取四个公司的股票数据
import pandas as pd
import numpy as np
from pandas_datareader import data

codes = ['AAPL', 'IBM', 'MSFT', 'GOOG']  #四个股票
all_stock = {}
for ticker in codes:
    all_stock[ticker] = data.get_data_yahoo(ticker)#默认从2010年1月起始, start
```

```
='7/1/2005

volume = pd.DataFrame({tic: data['Volume'] for tic, data in all_stock.iteritems()})
open = pd.DataFrame({tic: data['Open'] for tic, data in all_stock.iteritems()})
high = pd.DataFrame({tic: data['High'] for tic, data in all_stock.iteritems()})
low =  pd.DataFrame({tic: data['Low'] for tic, data in all_stock.iteritems()})
close =  pd.DataFrame({tic: data['Close'] for tic, data in all_stock.iteritems()})
price = pd.DataFrame({tic: data['Adj Close'] for tic, data in all_stock.iteritems()})
```

In [53]:
```
all_stock['AAPL'].head()
```

Out[53]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2010-01-04	213.429998	214.499996	212.380001	214.009998	123432400	27.727039
2010-01-05	214.599998	215.589994	213.249994	214.379993	150476200	27.774976
2010-01-06	214.379993	215.230000	210.750004	210.969995	138040000	27.333178
2010-01-07	211.750000	212.000006	209.050005	210.580000	119282800	27.282650
2010-01-08	210.299994	212.000006	209.060005	211.980005	111902700	27.464034

In [54]:
```
price.head()
```

Out[54]:

	AAPL	GOOG	IBM	MSFT
Date				
2010-01-04	27.727039	313.062468	111.405000	25.555485
2010-01-05	27.774976	311.683844	110.059232	25.563741
2010-01-06	27.333178	303.826685	109.344283	25.406859
2010-01-07	27.282650	296.753749	108.965786	25.142634
2010-01-08	27.464034	300.709808	110.059232	25.316031

In [55]:
```
all_stock['AAPL'].head()
```

Out[55]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2010-01-04	213.429998	214.499996	212.380001	214.009998	123432400	27.727039
2010-01-05	214.599998	215.589994	213.249994	214.379993	150476200	27.774976
2010-01-06	214.379993	215.230000	210.750004	210.969995	138040000	27.333178
2010-01-07	211.750000	212.000006	209.050005	210.580000	119282800	27.282650
2010-01-08	210.299994	212.000006	209.060005	211.980005	111902700	27.464034

In [56]:
```
AAPL = all_stock['AAPL']
len(AAPL)
AAPL.head()
```

Out[56]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2010-01-04	213.429998	214.499996	212.380001	214.009998	123432400	27.727039
2010-01-05	214.599998	215.589994	213.249994	214.379993	150476200	27.774976
2010-01-06	214.379993	215.230000	210.750004	210.969995	138040000	27.333178
2010-01-07	211.750000	212.000006	209.050005	210.580000	119282800	27.282650
2010-01-08	210.299994	212.000006	209.060005	211.980005	111902700	27.464034

In [57]:
```
#为了以后更容易导入数据，我们生成一个新的csv数据文本，并将所有数据行写入新文件
AAPL.to_csv('data/AAPL-0.csv')
```

In [58]:
```
#有时网络访问不好，因此读入已经保存的AAPL股票数据
import numpy as np
import pandas as pd
f = 'data/AAPL-0.csv'
data = pd.read_csv(f, index_col='Date')  #使用date列作为行索引
data.index = pd.to_datetime(data.index)  #将字符串索引转换成时间索引
AAPL = data
print len(AAPL)
AAPL.tail()
```

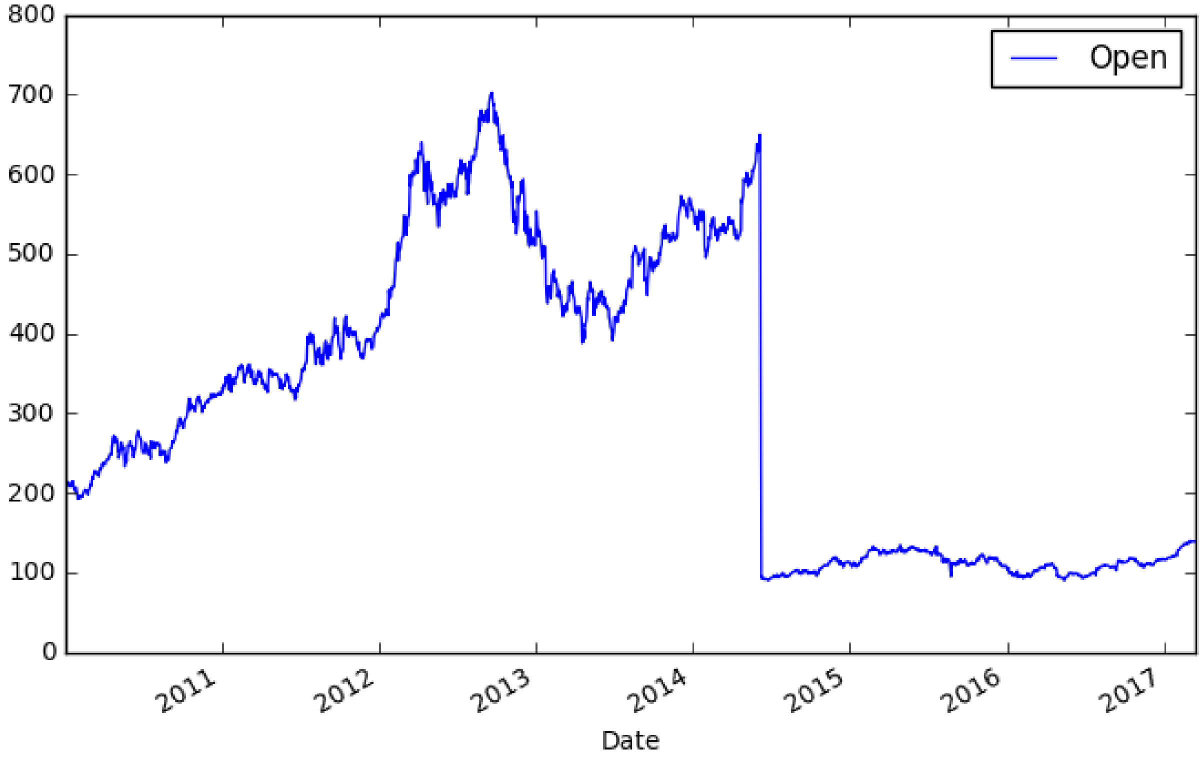
1812

Out[58]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-09	138.740005	138.789993	137.050003	138.679993	22065200	138.679993
2017-03-10	139.250000	139.360001	138.639999	139.139999	19488000	139.139999
2017-03-13	138.850006	139.429993	138.820007	139.199997	17042400	139.199997
2017-03-14	139.300003	139.649994	138.839996	138.990005	15189700	138.990005
2017-03-15	139.410004	140.750000	139.029999	140.460007	25566800	140.460007

In [59]:
```
#包含两种趋势的典型股价图表绘图
AAPL[['Open']].plot(figsize=(8,5))
```

Out[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x109b3978>
```



作业1-b：

- 从Yahoo！ Finance下载美交所中国各种题材股票（阿里，百度，京东等等）
- 从Yahoo！ Finance下载沪深交所各种题材股票（沪市SS，深市SZ）
- 分析并观察各种题材股票(保险类，新能源类，互联网相关)的各种统计情况、趋势、相关性分析等

In [60]:

```
# 使用Yahoo Finance的API获取沪深股市的股票数据
import pandas as pd
import numpy as np
from pandas_datareader import data

#获取
Maotai = data.get_data_yahoo('600519.SS') #茅台股票代码+沪市
Maotai.tail()
```

Out[60]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-09	367.00000	369.98001	365.56000	369.89999	2340600	369.89999
2017-03-10	369.92001	377.00000	368.35999	369.85001	4095100	369.85001
2017-03-13	370.04999	374.32999	367.53000	371.54999	2767400	371.54999
2017-03-14	371.54999	373.85001	368.34000	369.50000	2041600	369.50000
2017-03-15	369.50000	375.14999	369.01001	374.67999	2515500	374.67999

In [61]:

```
# 使用Yahoo Finance的API获取沪深股市的股票数据
import pandas as pd
import numpy as np
from pandas_datareader import data
```

```
#获取
PUFA = data.get_data_yahoo('600000.SS') #浦发银行股票代码600000+沪市SS
PUFA.tail()
```

Out[61]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-09	16.37	16.40	16.22	16.22	17366000	16.22
2017-03-10	16.23	16.28	16.17	16.23	16396300	16.23
2017-03-13	16.23	16.34	16.16	16.34	17950100	16.34
2017-03-14	16.34	16.35	16.24	16.26	16988900	16.26
2017-03-15	16.24	16.28	16.17	16.24	18900300	16.24

In [62]:

```
# 使用Yahoo Finance的API获取沪深股市的股票数据
import pandas as pd
import numpy as np
from pandas_datareader import data

#获取探路者股票代码是：300005，深市SZ
EXP = data.get_data_yahoo('300005.SZ') #探路者股票代码是：300005，创业板股票代码以300打头
EXP.tail()
```

Out[62]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-03-09	14.66	14.66	14.50	14.54	4817400	14.54
2017-03-10	14.58	14.61	14.42	14.45	4997900	14.45
2017-03-13	14.47	14.62	14.42	14.60	4972400	14.60
2017-03-14	14.58	14.69	14.44	14.69	5233300	14.69
2017-03-15	14.59	14.61	14.51	14.57	3791300	14.57